Demo Paper: Demonstrating Hybrid Provenance Queries from Script Runs

ABSTRACT

In this demo, we demonstrate prospective and retrospective provenances collected by YesWorkflow (YW) and DataONE RunManagers (R client and MATLAB client). For a workflow project, we have multiple provenance graphs consisting of a graph of prospective provenance, a graph of hybrid provenance, a graph of retrospective multi-run provenance. From our demonstration, we find that the demonstrated provenance capabilities can benefit earth researchers to understand the processing history of a derived products and verify them.

KEYWORDS

prospective provenance, hybrid provenance, retrospective provenance, hybrid provenance query, retrospective provenance query

ACM Reference format:

. 2016. Demo Paper: Demonstrating Hybrid Provenance Queries from Script Runs. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 6 pages.

DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Provenance from scripts and runs of scripts plays an important role in software debugging, testing, reliability and sharing. Such provenance traces consist of events that the user is interested in. A considerable amount of research has been done on investigating methods of harvesting provenance information from scripts and runs of scripts, ranging from conventional approaches, e.g. research compendium (folder layouts) and logging to recent provenance tools, e.g., YesWorkflow (YW) [ref], noWorkflow (NW) [ref], RDataTracker [ref], Reprozip [ref], RunManager [ref, ref]. However, most existing tools focus on provenance harvesting and not enough attention has been paid to study provenance queries systematically. The work is an attempt to fill the gap. On the other hand, DataONE is a federated data repository on the web providing access million of scientific datasets with structured science metadata and provenance data to facilitate not only searching of a dataset, but also the veracity or provenance of that dataset. This demo illustrates the prospective and retrospective provenance that are captured by YesWorkflow and DataONE RunManagers, and how to query, and visualize using a combination of tools for exposing both prospective, retrospective and hybrid provenance.

Conference'17, Washington, DC, USA

DOI: 10.1145/nnnnnnn.nnnnnnn

`Data provenance` is metadata that describes the origin and processing history of a data artifact. In the computational and data sciences, we can distinguish two forms of data provenance, i.e., `prospective provenance` and `retrospective provenance`. The former describes the general workflow by which data is produced, while the latter consists of runtime observables, e.g., names, locations, and contents of files read and written during a script run (coarse-grained observables). Fine-grained provenance can be captured as well, e.g., changes to individual data records, program variables, parameter settings, etc. `Hybrid provenance` can be defined as a graph containing an alternation of process steps, situated in the context of the entire process structure, and of the specific instances of data elements that have been generated and consumed by executions of those steps. The data elements can be instantiated to document the specific files used in a concrete workflow run.

In this demo, YW and DataONE RunManagers are used as examples.We show how prospective provenance can be expressed using YW annotations and how to use RunManagers for R and MATLAB to capture runtime file-level provenance information that are interested by the earth science community. Then, we show how to produce hybrid provenance by joining prospective and retrospective provenance with the YW URI mechanism. Furthermore, we use two real-life examples (OHI Howe Sound [ref] and Carbon 3/Carbon 4 soil mapping [ref]) as use cases.

Last but not least, we propose `multi-run provenance`. From the multi-run provenance, it enables a longitudinal view of a typical real-life scientific workflow that consists of multiple phases. Since computational and data science experiments can often last days, weeks, or even months and often require the execution of multiple scripts or workflows with varying input datasets and parameters, some of these script runs appear as chained together implicitly via intermediate data files, i.e., the output of one script run was subsequently used as an input to another script run. In this way, larger ad-hoc dataflow graphs are generated as part of a data-driven, exploratory workflow. By multi-run provenance we are referring to those dataflow graphs that link individual script runs via intermediate data files.

To summarize, we demonstrate the query-based provenance data dependency analysis with queries. A graph is used to represent the query results. For a workflow project, we have multiple provenance graphs consisting of a graph of prospective provenance, a graph of hybrid provenance, a graph of retrospective multi-run provenance.

(1) **Prospective provenance queries in the context of a single script.** This can expose and test data dependencies at the workflow-level.
(2) **Retrospective provenance queries in the context of a single run of a single script**: captures actual input and output files of a script run and other runtime observables.

(3) **Hybrid provenance query in the context of a single script and single run**: blends retrospective and prospective provenance, yielding new knowledge artifacts.
(4) **Provenance query in the context of multi-run provenance**: query and visualize data dependencies across multiple script runs.

The rest of this paper is organized as follows. The system architecture and key features of YW and RunManagers are presented in Section 2. Section 3 discusses two example use cases whereas Section 4 concludes the paper.

Related Work. Table 1 presents a representative set of related provenance tools for provenance collection. YesWorkflow provides prospective graph revealing the data dependencies at the conceptual workflow level. URI-template specifies a file path pattern while user-defined log-files capture runtime provenance observables at any level of granularity. The noWorkflow system (NW) [ref] captures code-level retrospective provenance by employing a Python profiling library to obtain Python function call chains and variable assignments. The DataONE RunManagers [ref] overload and thus intercept file I/O operations to automatically capture runtime observables at the file level.

2 PROVENANCE SYSTEM ARCHITECTURE

System architecture. Figure 1 shows provenance query architecture. After a user annotates a script with YW tags a script (in Python, R, MATLAB, ...), the dataflow graph implicit in the script can be revealed as a YW workflow model (left). System-recorded or user-provided runtime observables (right) can then be linked to the workflow model via suitable "bridge rules" (expressed e.g. as Prolog or SQL views), which in turn are used to generate new hybrid provenance graphs (center).There are three provenance sources: prospective, retrospective and hybrid provenance. Prospective provenance represents user-defined workflow models whereas retrospective provenance come from runtime observables. Hybrid provenance is a joint product of prospective- and retrospective provenance.

Prospective provenance tool (YesWorkflow) and retrospective provenance tool (RunManagers). YW and RunManagers that are used by DataONE provenance team are used for demo. YesWorkflow (YW) is a prospective provenance tool allowing users to easily recover high-level workflow models latent in scripts using simple user-annotations embedded as script comments [ref]. The workflow models declared by users and extracted via the YW toolkit can be augmented and enriched by retrospective provenance observables, yielding different forms of hybrid provenance [ref ...], i.e., whose "workflow backbone" is given by the user-declared YW model (prospective provenance) and whose execution details are filled in from one or more sources of runtime observables (retrospective provenance), cf. Figure 2.

Figure 2 shows a code snippet from a MATLAB script with YW annotations: Here, the @begin and @end tags are used to mark code blocks (i.e., processing steps in the YW model); @in and @out tags are used to define the dataflow between blocks; and the @uri template (which is paired with an @in tag) is used to define a where the data file is located. Note that @uri templates define a metadata pattern via so-called template variables (here: *start_year*, *end_year*,

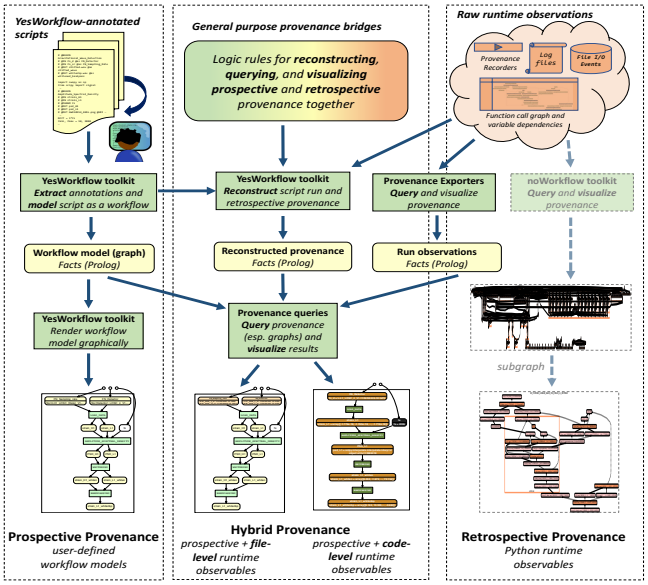


Figure 1: Provenance Query Architecture

```
%% Load input: long-term monthly mean air temperature data
% @@BEGIN fetch_monthly_mean_air_temperature_data
% @IN mean_airtemp
% @URI
file:inputs/narr_air.2m_monthly/air.2m_monthly_{start_year}_{end_year}_mean.{month}.nc
% @OUT Tair @AS Tair_Matrix
Tair=zeros(ncols,nrows,12);
for m=1:12
   tncid=netcdf.open(strcat('inputs/narr_air.2m_monthly/air.2m_monthly_2000_2010_mean.',
                            num2str(m),'.nc'),
                     'NC_NOWRITE');
   tvid = netcdf.inqVarID(tncid, 'Tair_monthly_mean');
   Tair(:,:,m) = netcdf.getVar(tncid,tvid);
   netcdf.close(tncid)
end
% @END fetch_monthly_mean_air_temperature_data
```

Figure 2: An example YW annotation block in the C3C4 script

and *month*). In this way, the script author can document the folder structure and file-naming conventions used by the script. When the script is run, concrete file- and folder-names are instantiated and can be harvested subsequently by YW to obtain runtime provenance information (even in the absence of a dedicated provenance recorder [ref]). When combining this retrospective provenance information with the YW workflow model, hybrid provenance queries can be answered.

RunManagers are DataONE provenance tools for capturing retrospective provenance from MATLAB and R script executions. The DataONE RunManagers overload and thus intercept file I/O operations to automatically capture runtime observables at the file level. DataONE provides two implementations: the recordr R package and the MATLAB toolbox. The provenance captured during a script execution includes information about the script that was run, the files that were read or written, and details about the execution environment at the time of execution.

The retrospective provenance tool (DataONE RunManagers) support the APIs for R and MATLAB users to capture, search, archive, and share a script run, such as, `record()`, `startRecord()`, `endRecord()`, `listRuns()`, `deleteRuns()`, `viewRun(s)()`,

Table 1: Various Provenance Tools and Methods Breakdown

	Language: Application / Implementation	Scope	Methods
YesWorkflow	Any / Java	Prospective Provenance	Annotations added by users; language independent
URI-templates	Any / Any	Retrospective Provenance	Templates matching with runtime file locations; language independent
User-defined log-files	Any / Any	User-defined Retrospective Provenance	Exported runtime variables and files provenance by users
noWorkflow	Python / Python	Retrospective Provenance	Runtime function calls, variables, file inputs/outputs, execution environmental information
RunManagers	R / R MATLAB / MATLAB	Retrospective Provenance	Runtime file inputs/outputs, execution environmental information

`traceRuns()` for R, `plotRuns()` for R and `publishRun()`. The captured retrospective provenance are populated to a relational provenance database which has these tables for `execution metadata`, `file metadata`, `tag`, `module dependencies` for MATLAB and `prov relationships` for R. Furthermore, these relational retrospective can be exported to Prolog facts and YAML formats. On the other hand, the captured provenance can be packaged and published to the DataONE member node federation network. A DataONE provenance package includes the script itself, input files, generated files that are associated with the run, science metadata file that documents the run and a resource map.

3 EXAMPLE USE CASES

In this demo, two real-life scientific examples are used: (1) a workflow for Ocean Health Index (OHI) for Howe Sound, British Columbia [ref]; (2) a workflow for Carbon3/Carbon4 (C3/C4) soil mapping for North America [ref]. The OHIBC Howe Sound workflow runs in a batch mode.

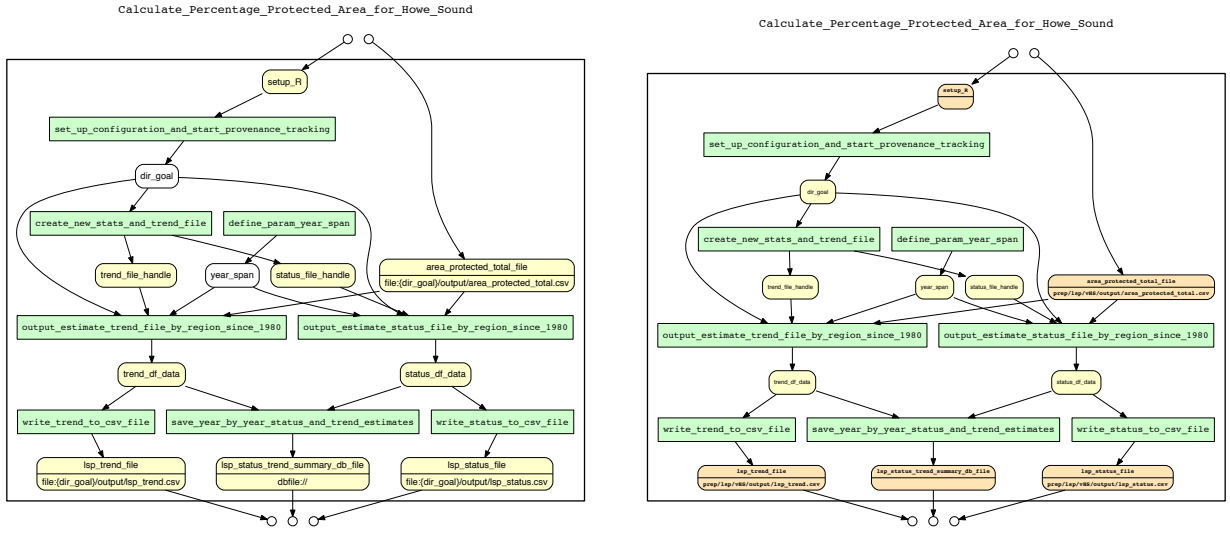
Based on RunManager R provenance database, Figure 3 gives us an overview for the OHIBC Howe Sound workflow example. The goal model for Howe Sound is: a region's status is based upon percent of protected area within 1 km inland buffer and percent of protected area within 3 nautical mile offshore buffer, compared to a reference point of 30% protected area. The data sources used in the OHIBC workflow include: (1) BC-specific WDPA dataset that was created from WDPA global dataset, then rasterize to BC Albers at 500m resolution; (2) BC Parks, Ecological Reserves, and Protected Areas (PEP) data. The OHIBC workflow creates two outputs: one file is for estimate of status by region since 1980 (`lsp_status.csv`) and the other file is for estimate of trend by region since 1980 (`lsp_trend.csv`). Figure 3 also shows the methods used in the OHIBC workflow example and each method name is represented using a script name. For example, the script `rasterize_HS_WDPA_and_PEP.R` rasterizes the BC WDPA-MPA shapefile to Howe Sound extents; the script `lsp_zonal_stats.R` calculates the goal model for Howe Sound explained above; and the script combines scores for inland and offshore and writes to output layers (`estimate_status_and_trend_by_year.R`).

To demonstrate the provenance data dependency analysis, we have developed SQL queries as well as rule-based queries (implemented in Prolog) against various provenance facts. A graph is used to represent the query results. For example: What outputs

Figure 3: Complete retrospective provenance graph for the OHI Howe Sound workflow example plotted by the API `plotRun()` in DataONE RunManager R client

does the workflow (script) have? Advanced provenance queries can take the form of regular path queries (RPQs), tree-structure queries, or graph queries and show the data lineage hidden in the provenance information. For example: (1) What are the ancestors or descendants of a given output in a prospective provenance graph? (2) What are the ancestors or descendants of a given output in a retrospective- or hybrid- provenance graph? (3) Which possible execution (i.e., dataflow) paths satisfy a given regular path expression? In this demo, we show each of these queries in action. The queries can be found at [ref1, ref2].

The first category provenance query is `prospective provenance queries` that refer to workflow steps and data names declared via YW annotations. A YW prospective provenance query can tell us if an output depends a particular input or if an output depends on all of inputs. Therefore, YW prospective provenance can add finer workflow model information for each script in order to share the relevant computational steps and data elements with others.



(a) label1 **(b) label2**

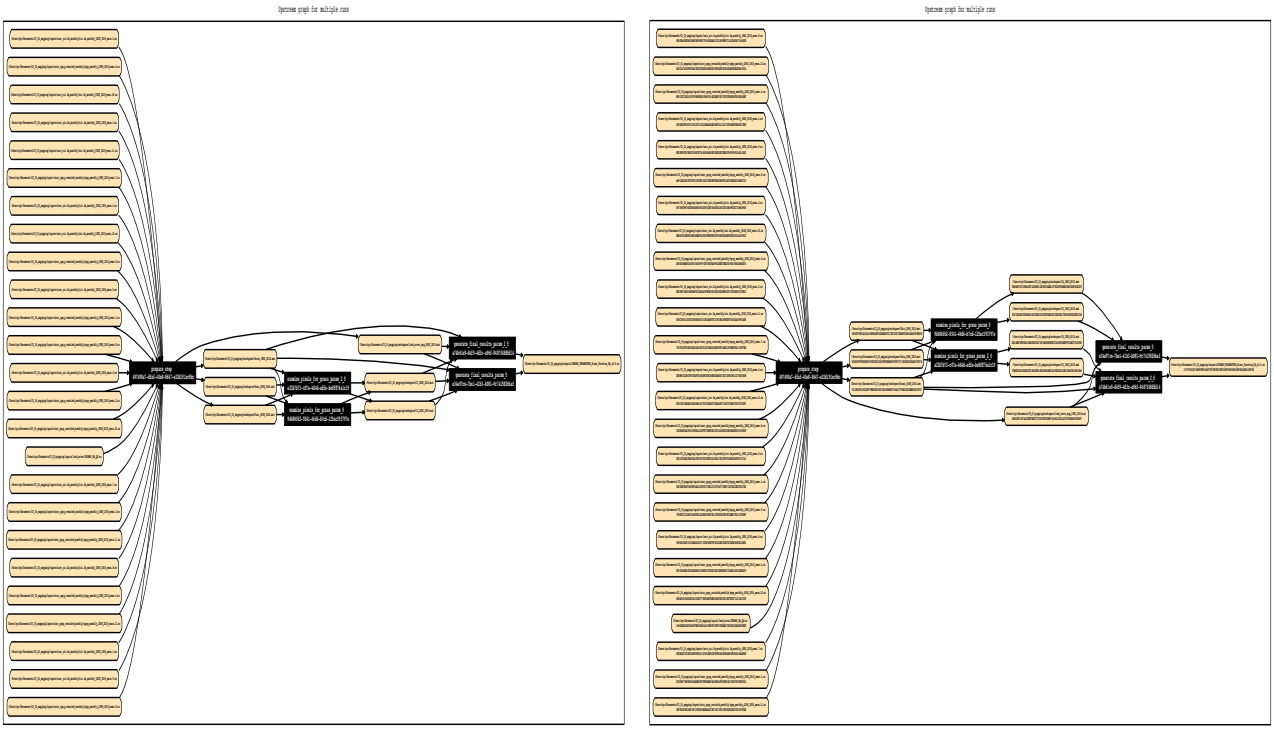
Figure 4: Complete YW prospective and hybrid provenance graphs for the OHIBC Howe Sound example. The diagram shows the script `estimate_status_and_trend_by_year.R` that calculates the "lasting special placessub goal" of the Ocean Health Index for Howe Sound, British Columbia that is based on the percentage of protected area within 3km of shore and 1km buffer zone inland, and how that has changed over time. The green box represents a computation step and a round yellow square box represents a data element declared in the script via YW tags.

For example, the script `estimate_status_and_trend_by_year.R` is used as an example (see Figure 4). The green box represents relevant computational steps, the yellow round box represents data elements (also called data flow) and the white round box represents parameters that are not consumed by computational steps. Hence, the prospective provenance added by YW is valuable for script-based scientific workflow provenance.

The second category provenance query is `hybrid provenance query`, that is querying YW*DataONE RunManager retrospective provenance. YW*DataONE RunManager hybrid provenance queries refer to YW model and runtime file-level retrospective provenance harvested by RunManager. In a hybrid provenance graph, the "workflow backbone" is given by the user-declared YW model (prospective provenance) and the execution details are filled in from one or more sources of runtime observables (retrospective provenance). Figure 4 shows a hybrid provenance graph for the last script in the OHIBC example `estimate_status_and_trend_by_year.R`. We can see that the yellow round box changes to orange round box and the file path pattern embedded is expanded with runtime relative file paths. This graph can tell us the runtime input files and output files. We have another example for Carbon3/Carbon 4 in which there are twelve files matched the YW URI template. Then the URI template is expanded to twelve files in the orange box so that a user can know the actual used file information.

The third category provenance query is `multi-run provenance query`, that is querying DataONE RunManager retrospective provenance. RunManager retrospective provenance queries refer to the runtime retrospective provenance obtained from executing multiple scripts, using RunManager. Since the scripts in a project are usually logically connected through sharing some of their inputs and outputs, the corresponding provenance traces are also connected through shared data products. One example is pictured in Figure 3 for the OHIBC workflow example. Besides this valuable overview graph, we provide other formats of multi-run provenance graphs for scientific workflow management: multi-run graph based on file path and multi-run graph based on hash code(see Figure 5).

The second example script (in MATLAB) produces Carbon 3/Carbon 4 (C3/C4) soil maps for North America using average rain and air temperature monthly data from year 2000 to 2010. In order to simulate real-life workflow, we create a version 2 C3/C4 example by splitting one script to three phases: preparing step, examining pixels for grass and generating final results. After the preparing step, a user can run the second script `examine pixels for grass` for multiple times with different parameters in order to find a proper solution, then writes to the output files. The generated graph recorded a set of runs that consist of the three scripts, five runs and the second script was run with two different parameter values (i.e., 2.5 and 5) twice. Figure 5 depicts two views of multi-run graphs. By comparing these two figures, we can understand that the second script which was run twice produces two versions of files but write to the same file locations.



(a) label 1

(b) label 2

Figure 5: Upstream retrospective provenance subgraph (longitudinal view) that is relevant to one output "SYNMAP_PRESENTVEG_Grass_Fraction_NA_v2.0.nc" for C3C4 version 2 example. The presented graph recorded a set of runs that consist of three scripts, five runs and the second script was run with two different parameter values (i.e., 2.5 and 5) twice. The example script set is called C3C4 example version 2. In the diagram, an orange yellow box represents a data file and a black box represents an execution (a run).

Reproducibility. DataONE allows to make experiments reproducible by providing two capabilities. First, as we have demonstrated above, DataONE RunManagers are able to capture both prospective and retrospective provenance (following the ProvONE model) by keeping tracking of the complete user-defined workflows implicit in a workflow script as well as the full history of the data file produced and consumed during each execution. Second, DataONE data repository allows users to package workflow results (implemented in any programming languages such as Python, R, MATLAB, SAS, ...) in a structured DataONE data package and publish to a member node inside a DataONE federation network so that the workflow results (including 2D graphs, 3D graphs, scripts, data, provenance relationship, science metadata describing the experimental methods, authors, geographical coverage, time coverage provided by data providers, ...) can be displayed on DataONE data repository and be shared or linked with other scientific experiments in a larger scale. It is different from providing reproducibility based on IPython notebook [ref] or a virtual machine, such as a Docker or a VM. Compared to myExperiment [ref], DataONE package contains provenance information (`prov:used`, `prov:wasGeneratedBy`, or `prov:wasDerivedFrom`) to describe the relationships between digital objects and datapackages for reproducibility, although both myExperiment and DataONE use package to encapsulate digital objects such as input data, final results and provenance.

4 CONCLUSIONS

Provenance is a useful component of the software development cycle, either to benefit researcher themselves or documenting and sharing code for use by future investigators. For the researcher, the quality of data produced by script runs can be assessed by identifying source data and verifying their quality.

Many provenance tools have been developed to capture, store, query and visualize the provenance produced by scripts and runs of scripts. But there is not enough investigation on what kind of provenance queries can be answered from the provenance information at prospective and retrospective provenance levels. Our research on prospective, hybrid and retrospective provenance queries allow researchers to use simple queries and a subset of advanced queries to query across YW model and various retrospective provenance sources to produce interesting provenance artifacts.

With our provenance query demo, we showcase our prospective, retrospective and hybrid query capabilities. We demonstrated that

provenance queries across heterogeneous provenance sources can yield a more complete and comprehensible picture of data provenance. We also think provenance describing datasets can facilitate data discovery for others.